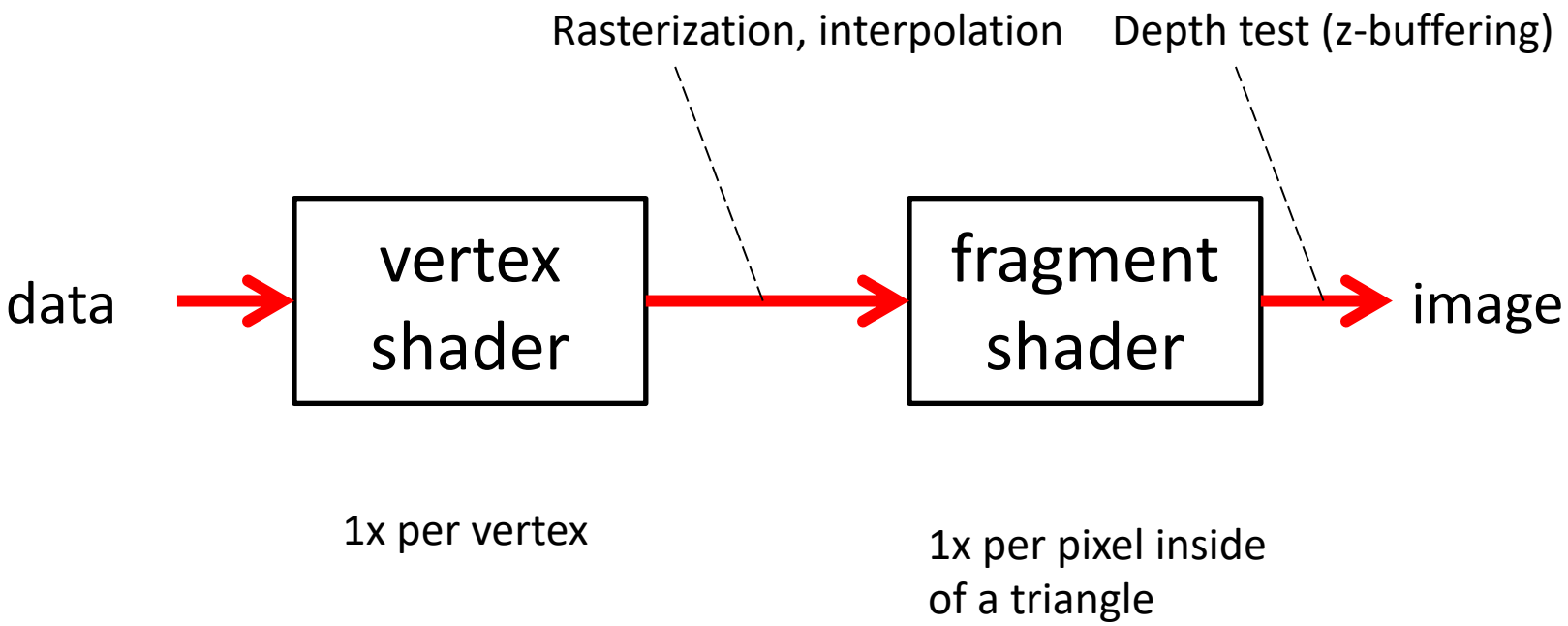
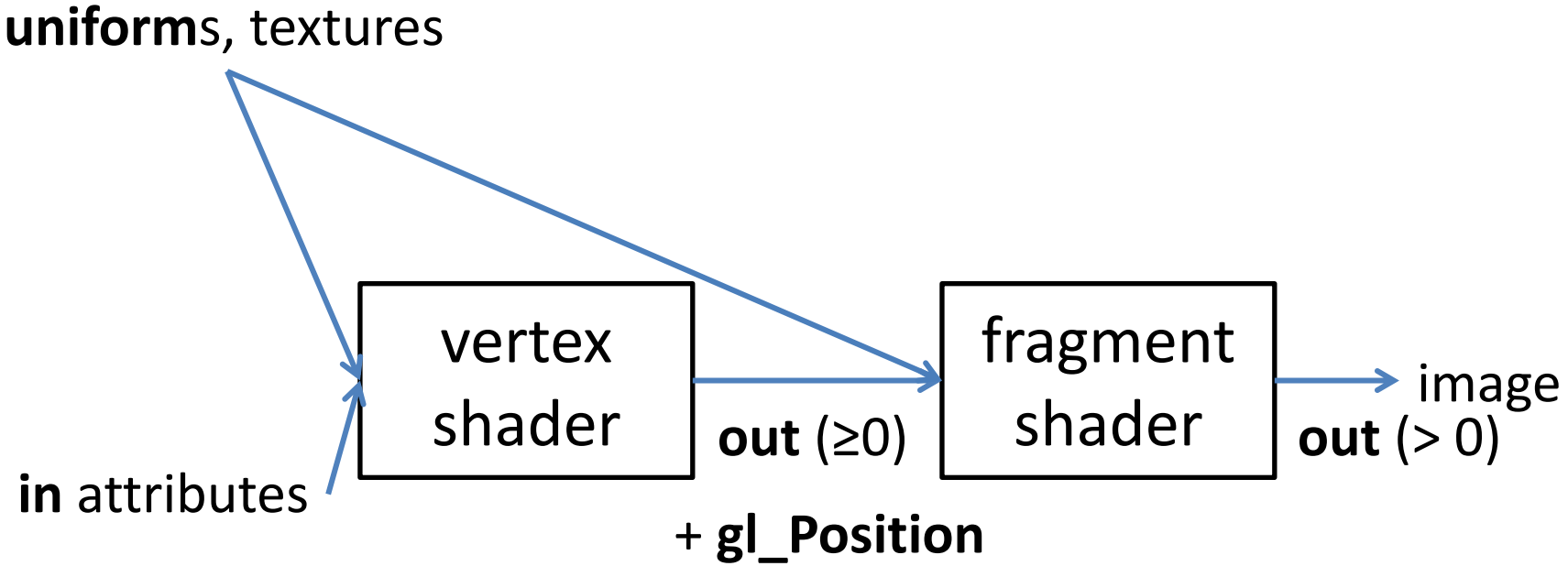
Project 3

Shading

GLSL Basics


Rasterization, interpolation
Depth test (z-buffering)
data
vertex shader
fragment shader
image
1x per vertex
1x per pixel inside of a triangle

GLSL Basics



You give *meaning* to any **in** attributes and **out** variables (except gl_Position and the final color going to the screen) as well as uniforms and 'textures'.

Vertex **in** Attribute examples: normals, colors, vertex-positions, uv etc. [read-only]

out vs -> ps examples: normals, colors, vertexpositions, uv, etc. [interpolated perspective-correctly, write-only]

uniform : properties of lightsources, transformation-matrices etc. [aka. constants]

GLSL

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

GLSL

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

GLSL

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

GLSL Datatypes

int, float, bool as in C/C++

vec2, vec3, vec4:

vectors of float elements

Addition is vector addition (no .add() call needed!), multiplication with matrices with just *.

GLSL Datatypes

vec2, vec3, vec4:

support **swizzling**: Direct access to x,y,z,w-components possible *in any order* by using point-operator „.“

Example:

```
vec4 myVec = vec4(1,2,3,4);
myVec.x      -> 1
myVec.xy     -> (1,2)
myVec.yx     -> (2,1)
myVec.zzz    -> (3,3,3)
myVec.wwxy   -> (4,4,1,2)
```

This can do more than attribute-access in most programming languages!

GLSL Datatypes

mat3, mat4:

Float-matrices of size 3x3 or 4x4

Access on specific element possible with double brackets „[ ][ ]“
Access on specific row possible with single bracket „[ ]“

GLSL Datatypes

mat3, mat4:

Float-matrices of size 3x3 or 4x4

Access on specific element possible with double brackets „[ ][ ]“
Access on specific row possible with single bracket „[ ]“

Example:

```
mat3 myMat= (1, 2, 3,
             4, 5, 6,
             7, 8, 9);
myMat[2][2]      -> 9
myMat[2]         -> (7,8,9)
```

GLSL Datatypes

sampler2D:

Identifier for 2D-texture

Used to fetch data from texture via texture()

…more on this later…

Example

diffuse shading with texture for *one* directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

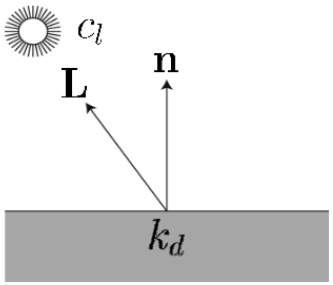
fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```



Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

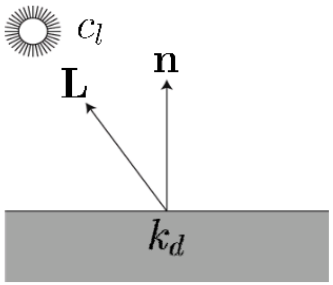
fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```



Important:
L and n have to be in the same coordinate system!

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

pass texture coordinate on to fragment shader

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

predefined output variable gl_Position
The vertexshader **must** always assign a value to this!

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

$$\mathbf{p}' = \mathbf{DPC}^{-1}\mathbf{Mp}$$

Object space

World space

Camera space

Canonic view volume

Image space

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

***texture*(*sampler2D*, *vec2*)**
= predefined function for texture-fetches.

argument 1: texture identifier
argument 2: (u,v) - texture coordinates
(normalized to range [0,1])

Return value is of type ***vec4***

Functions

- You can write functions as you would in C or in Java with **public static**
- They can have **out** and **inout** arguments, which behave like references (to objects)
- This allows for multiple return values.

```
int newFunction(in bvec4 aBvec4,   // read-only
                out vec3 aVec3,    // write-only
                inout int aInt)   // read-write
{
// do something interesting, must assign to aVec3…
return 0;
}
```

Passing Variables from Host to GLSL

- How do we set up the **uniform** variables?

- How about the **in** per vertex attributes for the vertex shader?
  - This is being done for you in jrtr, check out what happens with the SEMANTICS that you pass when creating VertexData objects.
  - Not covered here.

Passing Variables from Host to GLSL

Uniforms:

glUniform* binds uniform data to names in shader

Each datatype has his own **glUniform***-function:

glUniform(1|2|3|4)(f|i):

(1|2|3|4): dimension of type
(f|i): type (float or int)

Passing Variables from Host to GLSL

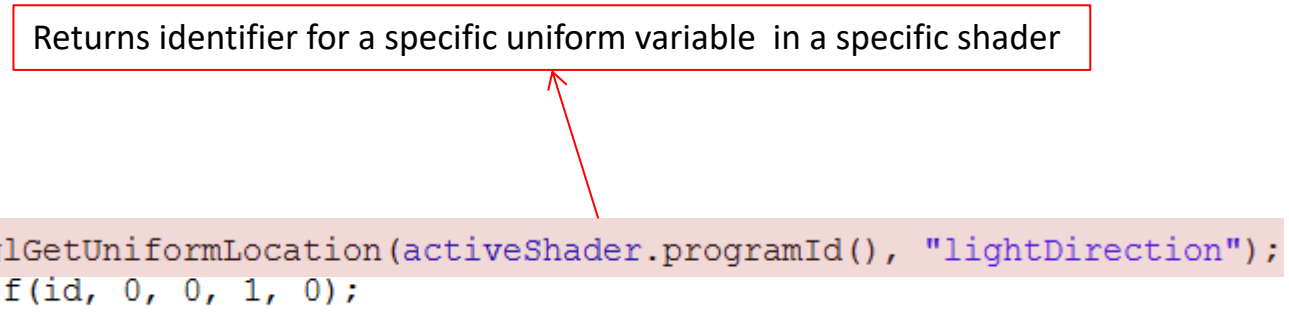
Example:

```
int id = gl.glGetUniformLocation(activeShader.programId(), "lightDirection");
gl.glUniform4f(id, 0, 0, 1, 0);
```

"gl." is not part of the name of this function (in official OpenGL documentations), but this is what you get for wrapping a procedural interface in an OO language…

Passing Variables from Host to GLSL

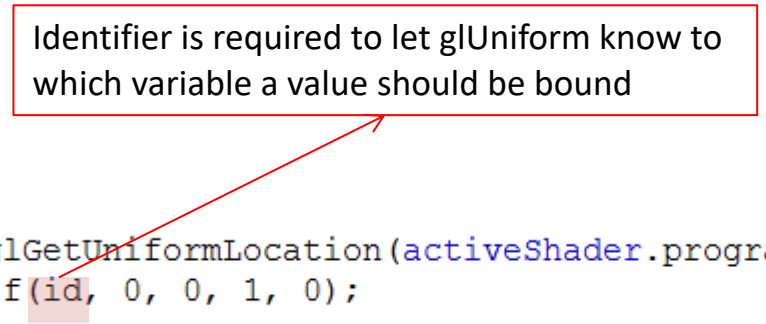
Example:




```
int id = gl.glGetUniformLocation(activeShader.programId(), "lightDirection");
gl.glUniform4f(id, 0, 0, 1, 0);
```

Passing Variables from Host to GLSL

Example:




```
int id = gl.glGetUniformLocation(activeShader.programId(), "lightDirection");
gl.glUniform4f(id, 0, 0, 1, 0);
```

Passing Variables from Host to GLSL

Example:

Type of univorm variable in shader
Here its **vec4**

```
int id = gl.glGetUniformLocation(activeShader.programId(), "lightDirection");
gl.glUniform4f(id, 0, 0, 1, 0);
```

Passing Variables from Host to GLSL

Example:

Data that should be passed. List of arguments depends on type. Here, for vec4, we need 4 values

```
int id = gl.glGetUniformLocation(activeShader.programId(), "lightDirection");
gl.glUniform4f(id, 0, 0, 1, 0);
```

Passing Variables from Host to GLSL

Arrays are passed with *glUniform****v*** (more on this later...)

Matrices are passed with *glUniform****Matrix*********v***

For more detail refer to:
https://www.opengl.org/sdk/docs/man/html/glUniform.xhtml

Assignments

Write some interesting per-vertex and per-pixel programs aka. Shaders in GLSL.

Together with the Java-side code needed to pass data to them.

Assignments

Preparation:

- Study how jrtr manages «materials» and «light sources» (refer to assignment description):
  - SceneManager stores lightsources in a list
  - Shapes have a reference to a material
  - Materials have a reference to the shaders they use
  - glRenderContext sets up a material for rendering by activating its shader and passing on uniform variables

Note that OpenGL knows nothing about ‘materials’ and ‘light sources’ (this used to be different).

Assignments

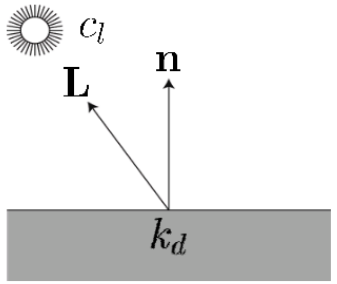
1. Create a diffuse shader for multiple point lights:

   - Point lights have a radiance c_l and a position p.
   - Objects have a diffuse reflection coefficient k_d

   Uniforms!

   - **Diffuse shading of an object with several point lights is:**

$$\sum_i c_{l_i} k_d (n \cdot L_i)$$



Assignments

1. Create a diffuse shader for multiple point lights:

   - Uniforms of the point lights can be passed as array
   - **But**: Size of arrays must be known to GLSL ***at compile time!***
   - Therefore its okay if the maximum number of point sizes is fixed (as long as its >1 ☺)

```
#define MAX_LIGHTS 8

uniform vec3 light_color[MAX_LIGHTS];
```

Assignments

1. Create a diffuse shader for multiple point lights:

   - To pass arrays we need to use **glUniform*v**

     Example:

```
int numOfElements = 3;
float[] dataArray = {1,2,3,4,5,5};
int offset = 0;

gl.glUniform2fv(id, numOfElements, dataArray, offset);
```

     Then the uniform-array in the shader has the values

     { vec2(1,2), vec2(3,4), vec2(5,5) }

Assignments

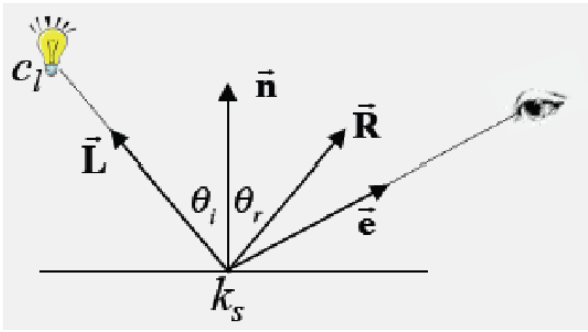
2. Per-pixel Phong shading for several point lights:

- Objects have an additional specular reflection coefficient k_s and a Phong-exponent p.

  => more Uniforms!

- **We need to calculate:**

$$\sum_i c_{l_i} \left(k_d (n \cdot L_i) + k_s (R \cdot e)^p \right)$$



- **R** can be computed using the predefined function **reflect**
- For computing **e** its useful to pass the camera position as uniform variable, because $e = camera_{pos} - surface_{pos}$

Assignments

3. Texturing:

- Copy & modify shader from exercise 2 to support textures!
  - Meaning k_d becomes the color that is fetched from the texture

- Further, copy & extend the shader to support a **gloss map:**

Assignments

3. Texturing:

- **Gloss map**:

  An additional texture whose brightness (sum of R,B and G) is used to control the specular coefficient k_s.

You can also use the alpha channel of an existing texture or so...

Assigmnent

4. Experiment with shaders:

Create your own shader, that can do whatever you want ☺

Assigmnent

4. Experiment with shaders:

You can use code from the internet if you want!

(but you probably need to modify it to work with jrtr...)

...but feel free to do your own stuff!

Shader Ideas

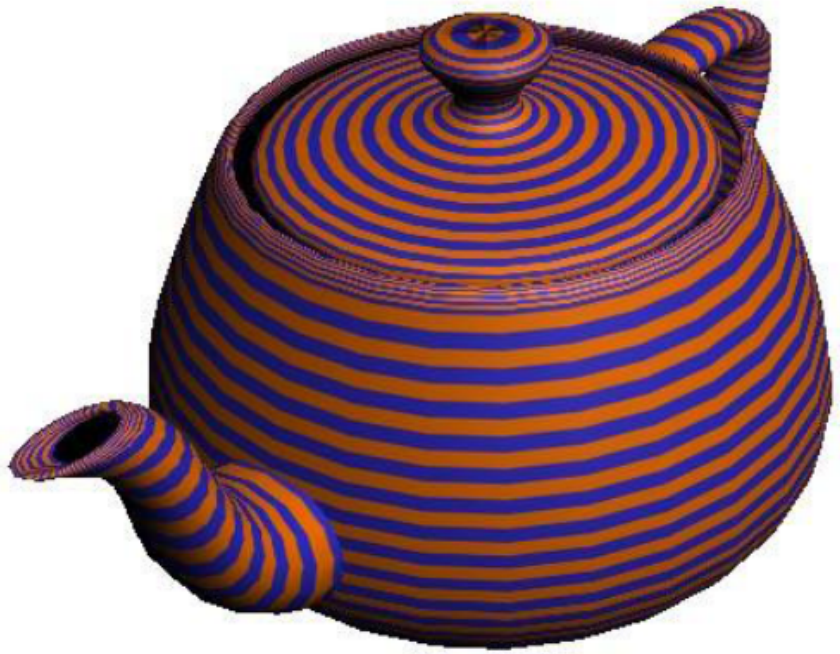
Toonshader

Shader Ideas

Procedural Brick Shader

Shader Ideas

Procedural Stripe Shader



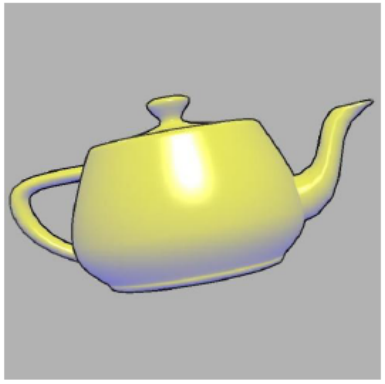
Shader Ideas

Procedural Noise Shader

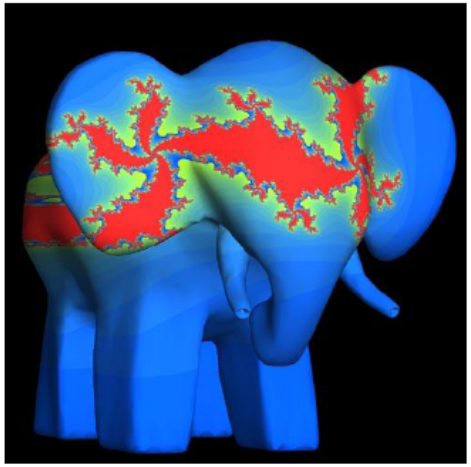
(if you dare...)

Shader Ideas

Glyph Bombing

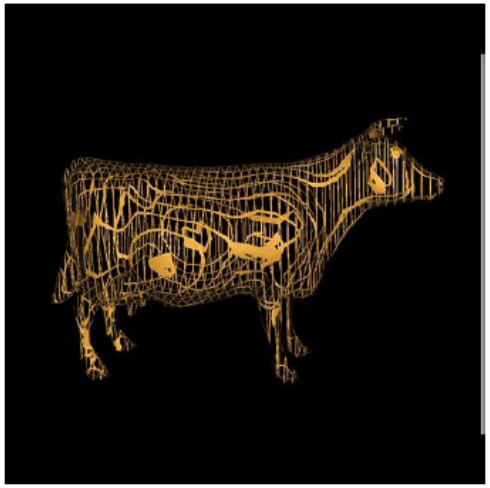


Gooch Shading



Julia Set

Toy Ball



Discarding Fragments

Shader Ideas

- Many more on ShaderToy (this is an art form!)

Remarks

Get some tools for GLSL authoring (at least syntax highlighting).

There is probably some plug in for your IDE.

Some GLSL features might be version-, platform-, graphics card- & driver- etc. – specific, watch out.